CleanOS: Limiting Mobile Data Exposure with Idle Eviction

Yang Tang, Phillip Ames, Sravan Bhamidipati, Ashish Bijlani, Roxana Geambasu, Nikhil Sarda
Columbia University
The Transition to Mobile Devices

- Mobiles are replacing desktops as primary computing platform

Desksops

- ✔ Linux, OS X, Windows
- ✔ Emails, Web browsers, Office

Mobile Devices

- ✔ Android, iOS, Windows Phone
- ✔ Emails, Web browsers, G Docs
- + New, fun, mobile apps
- + Increased productivity
- + Pervasive connectivity
Mobiles Bring New Challenges

Desksops

Physical security, firewalled networks

Mobile Devices

Unprotected outdoors, mobiles can be easily stolen, seized, or lost
OSes Not Designed for Challenges

• Despite threats, mobile OSes have not evolved to protect sensitive data

• Like desktop OSes, mobile OSes accumulate enormous sensitive data
  • OS doesn’t securely erase deallocated data
  • FS doesn’t securely erase deleted files
  • Applications hoard sensitive data for performance or convenience

• This data is placed at risk if device is stolen or lost
  • Thief can dump RAM, flash memory contents
  • Thief can break passwords
  • …
Examples of Exposed Data

<table>
<thead>
<tr>
<th>App</th>
<th>Password</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>Email</td>
<td>✔️</td>
<td>✔️</td>
</tr>
<tr>
<td>Y! Mail</td>
<td>✔️</td>
<td>✔️</td>
</tr>
<tr>
<td>Google Docs</td>
<td></td>
<td>✔️</td>
</tr>
<tr>
<td>OI Notepad</td>
<td></td>
<td>✔️</td>
</tr>
<tr>
<td>Dropbox</td>
<td></td>
<td>✔️</td>
</tr>
<tr>
<td>KeePass</td>
<td>✔️</td>
<td>✔️</td>
</tr>
<tr>
<td>Keeper</td>
<td>✔️</td>
<td>✔️</td>
</tr>
<tr>
<td>Amazon</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Pageonce</td>
<td>✔️</td>
<td>✔️</td>
</tr>
<tr>
<td>Mint</td>
<td></td>
<td>✔️</td>
</tr>
<tr>
<td>Google+</td>
<td></td>
<td>✔️</td>
</tr>
<tr>
<td>Facebook</td>
<td></td>
<td>✔️</td>
</tr>
<tr>
<td>LinkedIn</td>
<td></td>
<td>✔️</td>
</tr>
<tr>
<td><strong>TOTAL</strong></td>
<td><strong>5/14</strong></td>
<td><strong>13/14</strong></td>
</tr>
</tbody>
</table>

- We dumped memory and SQLite DB for 14 popular Android apps

**Default Email App:**

- Password and email snippets are kept in cleartext RAM at all times
- Everything is present in cleartext SQLite DB
Examples of Exposed Data

<table>
<thead>
<tr>
<th>App</th>
<th>Password</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>Email</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>Gmail</td>
<td></td>
<td>✔</td>
</tr>
<tr>
<td>Y! Mail</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>Google Docs</td>
<td></td>
<td>✔</td>
</tr>
<tr>
<td>OI Notepad</td>
<td></td>
<td>✔</td>
</tr>
<tr>
<td>Dropbox</td>
<td></td>
<td>✔</td>
</tr>
<tr>
<td>KeePass</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>Keeper</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>Amazon</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Pageonce</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>Mint</td>
<td></td>
<td>✔</td>
</tr>
<tr>
<td>Google+</td>
<td></td>
<td>✔</td>
</tr>
<tr>
<td>Facebook</td>
<td></td>
<td>✔</td>
</tr>
</tbody>
</table>

**TOTAL**  
5/14  
13/14

- We dumped memory and SQLite DB for 14 popular Android apps

**Overall:**

- Captured some sensitive data from 13/14 apps
- All 13/14 apps hoard some sensitive data in cleartext DB/RAM
- 9/14 apps hoard sensitive data in cleartext RAM at all times
Securing Data Is Darn Hard!

• Example protection systems:
  • Encrypted file systems
  • Encrypted RAM
  • Remote wipe-out systems

• Challenges / limitations:
  • Users don’t lock their devices (57%) or configure poor passwords
  • Physical attacks are notoriously difficult to protect against
    • E.g., memory dumps, cold boot attacks, breaking trusted-hardware
      seals can reveal data or decryption keys

• In general, these are imperfect stop-gaps on top OSes that were never designed with physical insecurity in mind
Time for New OS Abstractions

• Mobile OSes should manage sensitive data rigorously, so as to maintain the device “clean” at any point in time in anticipation of device theft/loss

• If device is stolen or lost:
  1. The minimal amount of sensitive data is exposed
  2. User retains post-theft control over unexposed data
CleanOS

- New Android-based OS that minimizes sensitive data exposure by **evicting it to a trusted cloud whenever not under active use**

- Implements **sensitive data objects (SDOs)**
  - Identifies locations of sensitive data in RAM and stable storage
  - Monitors its use by applications
  - “Evicts” sensitive data to the cloud whenever it is not under active use

- The cloud intermediates all accesses to unused SDOs and can offer a lot of **useful post-loss functions**:
  - Disable SDO access after theft
  - Audit SDO exposure and access
  - Rate-limit SDO accesses
This Talk

- All layers of current OSes require cleanup
- As first step, we focus on cleaning up the application layer
- This talk: limit exposure of sensitive data hoarded by apps
Outline

• Motivation
• Architecture and Prototype
• Evaluation
• Conclusions
Mobile OS Insights

1. Although exposed permanently, much sensitive data is only actually used very rarely
   - Email password is constantly exposed, but only used upon refresh
   - Subjects are constantly exposed, but only used upon inbox listing
   - Contents are often exposed, but only used when user reads email

2. Most mobile apps have a cloud component, which already stores the data

3. Mobiles are (becoming) mostly connected
   - Wi-Fi / cellular coverage is becoming pervasive
CleanOS Basic Functioning

- Applications **create SDOs** and **add sensitive data** to them
- CleanOS implements three **functions** for SDOs:
  1. **Tracks** data in SDOs using taint tracking
  2. **Evicts** SDOs to a trusted cloud whenever **idle**
  3. **Decrypts** SDO data when it is accessed again

---

When user reads an email:

<table>
<thead>
<tr>
<th>Email App</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Content SDO</strong></td>
</tr>
<tr>
<td><img src="image1.png" alt="Image" /></td>
</tr>
</tbody>
</table>

When the app goes to the background:

<table>
<thead>
<tr>
<th>Email App</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Content SDO</strong></td>
</tr>
<tr>
<td><img src="image3.png" alt="Image" /></td>
</tr>
</tbody>
</table>

evicted
CleanOS Increases Post-Loss Control

**Device lost/stolen**

- **Attacker**
  - No access to device or cloud keys

- **CleanOS**
  - Minimize data on the device

**User notices loss**

- **Tamper with device physically or in software:**
  - dump RAM
  - cold boot attacks
  - disable connectivity
  - break user password
  - decrypt all SDOs
  - …

- **CleanOS**
  - Control data accesses:
    - audit accesses
    - disable or rate-limit suspicious accesses
    - disable all accesses
The SDO Abstraction

- An SDO is a logical collection of Java objects that contain sensitive data and that are related somehow
  - Email-password SDO: password string + all objects computed from it
  - Email-contents SDO: all emails in a thread
  - Bank-account SDO: all transactions in an account

```java
class SDO {
    SDO(String description, ...); // create SDO
    void add(Object o); // adds object to SDO
    void remove(Object o); // removes object from SDO
    ...
    private int sdoID; // unique ID for SDO
}
```

human-readable and used for auditing

used for identifying SDOs locally and remotely
Example: A Clean Email App

- Default Email app hoards passwords and subjects
- Hard for apps to manage sensitive on their own
- With CleanOS, developers simply create SDOs and place their sensitive data in them

```java
SDO pwSDO = new SDO("Password of " + user);
pwSDO.add(password);
...
SDO emailSDO = new SDO("Email: " + mSubject);
emailSDO.add(mSubject);
emailSDO.add(mTextContent);
emailSDO.add(mHtmlContent);
emailSDO.add(mTextReply);
emailSDO.add(mHtmlReply);
```

(code adapted for clarity)
The CleanOS Architecture

Mobile Device

Application (Java)

Dalvik VM

Linux
The CleanOS Architecture

Mobile Device

Application (Java)

pwSDO

(SDO ID: 123, Key: 0x1a2b...)

Tracking (modified TaintDroid)

Eviction (evict-idle GC)

Decryption (modified interpreter)

CleanOS Dalvik VM

Linux

Trusted Cloud(s)

SDO Database

ID: 123
App: com.android.email
Desc: Password of Yang
Key: 0x1a2b...

Audit Log

Create SDO 123
App: ... Desc: ...
The CleanOS Architecture

CleanOS Dalvik VM

Mobile Device

Application (Java)

pwSDO

(SDO ID: 123, Key: 0x1a2b…)

Tracking (modified TaintDroid)

Eviction (evict-idle GC)

Decryption (modified interpreter)

Linux

Trusted Cloud(s)

SDO Database

pwSDO: Password of Yang

password

imap://user:password@imap.server.com:143

smtp://user:password@smtp.server.com:25

SDO ID: 123
App: com.android.email
Desc: Password of Yang
Key: 0x1a2b…

Audit Log

Create SDO 123
App: … Desc: …
The CleanOS Architecture

Mobile Device

Application (Java)

pwSDO

CleanOS Dalvik VM

Linux

Trusted Cloud(s)

SDO Database

<table>
<thead>
<tr>
<th>ID: 123</th>
</tr>
</thead>
<tbody>
<tr>
<td>App: com.android.email</td>
</tr>
<tr>
<td>Desc: Password of Yang</td>
</tr>
<tr>
<td>Key: 0x1a2b…</td>
</tr>
</tbody>
</table>

Audit Log

Create SDO 123
App: … Desc: …

(SDO ID: 123, Key: 0x1a2b…)

Tracking (modified TaintDroid)
Eviction (evict-idle GC)
Decryption (modified interpreter)
The CleanOS Architecture

Mobile Device

Application (Java)

pwSDO

CleanOS Dalvik VM

Linux

Trusted Cloud(s)

SDO Database

| ID: 123  |
| App: com.android.email |
| Desc: Password of Yang |
| Key: 0x1a2b… |

Audit Log

| Create SDO 123  |
| App: …  Desc: … |
| SDO 123 Evicted  |
| App: …  Desc: … |

CleanOS Dalvik VM

Tracking (modified TaintDroid)

Eviction (evict-idle GC)

Decryption (modified interpreter)

(SDO ID: 123, Key: Evicted)

sdoEvicted("com.android.email", 123)
The CleanOS Architecture

CleanOS Dalvik VM

Mobile Device

Application (Java)

pwSDO

SDO Database

ID: 123
App: com.android.email
Desc: Password of Yang
Key: 0x1a2b

Audit Log

Create SDO 123
App: ... Desc: ...

SDO 123 Evicted
App: ... Desc: ...

Fetch Key for SDO 123
App: ... Desc: ...

Trusted Cloud(s)

fetchKey("com.android.email", 123)

0x1a2b...

(SDO ID: 123, Key: 0x1a2b...)

Tracking (modified TaintDroid)

Eviction (evict-idle GC)

Decryption (modified interpreter)
SDO Tracking

- CleanOS uses a modified TaintDroid version to track SDOs
- TaintDroid uses 32-bit shadow taint tags to mark objects/primitives with one or more taints

\[
\begin{array}{c|c}
31 & \text{taint tag} \\
\hline
31 & 0 \\
\end{array}
\]

<table>
<thead>
<tr>
<th>Java object / primitive</th>
<th>000000000000000010000010</th>
</tr>
</thead>
</table>

- CleanOS imposes a **specific structure on taint tags**
  - Includes new tainting scheme to propagate one SDO ID (see paper)

\[
\begin{array}{c|c|c}
31 & \text{SDO ID} \\
\hline
31 & E \\
\end{array}
\]

<table>
<thead>
<tr>
<th>Java object / primitive</th>
<th>E</th>
</tr>
</thead>
</table>
Evict-Idle Garbage Collection

- To evict SDOs, we introduce a new type of Java garbage collector: the evict-idle garbage collector (eiGC)

- A traditional GC: deallocates only objects guaranteed to never be used in the future

- The eiGC: evicts objects that have not been used for a period of time, *even if* they might be used again in the future

- We run the eiGC periodically to evict idle SDOs
  - For any Java object that is tainted with an idle-SDO’s ID, the eiGC “evicts” that object, setting its E bit in the tag
Sensitive Data Encryption

• To “evict” a Java object, eiGC traverses it down to the data-bearing fields (primitives & arrays of primitives) and encrypt those with a key generated from the SDO’s key

Primitives:

Arrays of primitives:

```java
struct ArrayObject {
    contents[]; // evict in place
    counter;
    evicted_lock;
    ...;
}
```
CleanOS Defaults

• We hope for, but don’t rely upon app modifications

• Hence, we provide a set of solid defaults:
  • “SSL” SDO
  • “User Input” SDO
  • “Password” SDO

• Although coarse-grained, default SDOs limit exposure of sensitive data in unmodified apps

In SSL library (Java SDK):

```java
SDO sslSDO = new SDO("ssl");
sslSDO.add(cleartextDataBuffer);
```

In InputMethod (Android SDK):

```java
SDO inputSDO = new SDO("user input");
SDO pwSDO = new SDO("password");
if (type == InputType.TEXT_PASSWORD)
    pwSDO.add(text);
else
    inputSDO.add(text);
```

(code adapted for clarity)
Optimizations

CleanOS includes a number of optimizations, which help with both performance and energy consumption

1. **Pausing the eiGC**
   - After the app becomes idle, we stop running eiGC, until the app is active again

2. **Bulk eviction**
   - The eiGC evicts all SDOs at once soon after the app becomes idle

3. **Bulk prefetching**
   - The eiGC remembers which SDO keys were used recently and fetches all those keys at once
Prototype

• Modified Android and TaintDroid in significant ways

<table>
<thead>
<tr>
<th>Component</th>
<th>New or Changed Features</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dalvik (JVM)</td>
<td>Evict-idle Garbage Collector (eiGC)</td>
</tr>
<tr>
<td></td>
<td>Eviction-aware bytecode interpretation</td>
</tr>
<tr>
<td></td>
<td>Secure deallocation of interpreted stacks</td>
</tr>
<tr>
<td>Android SDK</td>
<td>SDO API</td>
</tr>
<tr>
<td></td>
<td>Default SDO heuristics</td>
</tr>
<tr>
<td>TaintDroid</td>
<td>Support for millions of taints</td>
</tr>
<tr>
<td></td>
<td>SQLite vulnerability fix</td>
</tr>
<tr>
<td>SQLite</td>
<td>Taint tracking in database</td>
</tr>
<tr>
<td>Webkit</td>
<td>Screen-buffer purging</td>
</tr>
<tr>
<td>Bionic (libc)</td>
<td>Secure user-space deallocation</td>
</tr>
<tr>
<td>Linux Kernel</td>
<td>Secure page deallocation with grsecurity</td>
</tr>
</tbody>
</table>

• Modified two “dirty” apps to use SDOs: Email and KeePass
  • Modifications are trivial: 7 LoC for each application
Post-Loss Services

• Built a CleanOS service on Google App Engine that supports post-loss auditing

<table>
<thead>
<tr>
<th>device</th>
<th>message</th>
<th>time</th>
</tr>
</thead>
<tbody>
<tr>
<td>cd5493c1befe9075442862afa046182</td>
<td>fetchKey(9.408 - com.android.email - password)</td>
<td>2012-04-28 17:26:50.590000</td>
</tr>
<tr>
<td>cd5493c1befe9075442862afa046182</td>
<td>registerSDO(com.android.email - Invitation to develop &quot;Clean OS&quot;)</td>
<td>2012-04-28 17:27:01.140000</td>
</tr>
<tr>
<td>cd5493c1befe9075442862afa046182</td>
<td>fetchKey(30.709 - com.android.keepass - Entry)</td>
<td>2012-04-28 17:27:48.500000</td>
</tr>
</tbody>
</table>

Screenshot of the CleanOS audit log

• Other services could be supported:
  • **Disable** keys after user notices theft
  • Monitor key accesses, detect anomalies, and **rate-limit/disable access**
  • Disable accesses to certain SDOs while **outside a trusted network**
  • …
Outline

• Motivation and Goals
• Architecture and Prototype
• Evaluation
• Conclusion
Questions

• Does CleanOS limit data exposure?
• Does CleanOS support effective auditing?
• Is CleanOS’ performance practical?
• Is CleanOS’ energy consumption practical?
• Is CleanOS network consumption practical?
Does CleanOS Limit Data Exposure?

<table>
<thead>
<tr>
<th>Email App Version</th>
<th>Password SDO</th>
<th>Contents SDO</th>
</tr>
</thead>
<tbody>
<tr>
<td>Email without CleanOS</td>
<td>100%</td>
<td>95.5%</td>
</tr>
<tr>
<td>Email with default SDOs</td>
<td>6.5%</td>
<td>5.9%</td>
</tr>
<tr>
<td>Cleaned Email with app SDOs</td>
<td>6.5%</td>
<td>0.3%</td>
</tr>
</tbody>
</table>

Fraction of time in which sensitive data was exposed

- Default SDOs reduce exposure by 89.6-93.5%
- App SDOs further reduce content exposure to 0.3%
- Similar results in Facebook and Mint apps

Thus, default SDOs drastically curb sensitive-data exposure
Is Auditing Effective?

Thus, for best auditing properties, applications should define their own SDOs.
Is CleanOS Practical (Wi-Fi)?

Thus, overheads are largely unnoticeable over Wi-Fi
Is CleanOS Practical (3G)?

![Graph showing performance comparison between Android, CleanOS, and Optimized CleanOS](image)

- **Email launch:**
  - Android: 722ms
  - CleanOS: 953ms
  - Optimized CleanOS: 354ms

- **Email read message:**
  - Android: 354ms
  - CleanOS: 354ms
  - Optimized CleanOS: 92ms

- **KeePass launch:**
  - Android: 1819ms
  - CleanOS: 1819ms
  - Optimized CleanOS: 1819ms

- **KeePass read entry:**
  - Android: 1819ms
  - CleanOS: 1819ms
  - Optimized CleanOS: 1819ms

- **Browser launch:**
  - Android: 1819ms
  - CleanOS: 1819ms
  - Optimized CleanOS: 1819ms

- **Browser load GNews:**
  - Android: 1819ms
  - CleanOS: 1819ms
  - Optimized CleanOS: 1819ms
Is CleanOS Practical (3G)?

Optimizations improve 3G performance significantly, making CleanOS practical.
Conclusions

• Mobile OSes and apps accumulate a lot of sensitive data, exposing it to attacks if device is stolen or lost

• CleanOS is a new Android-based OS design that minimizes amount of sensitive data exposed on device at any time
  - SDO: new OS abstraction for managing sensitive data
  - eiGC: garbage-collector that evicts unused SDO data

• CleanOS brings new view on data security: minimize and audit exposure of sensitive data to attack
  - We believe that this view is applicable more broadly to other domains, such as datacenter and Web data security